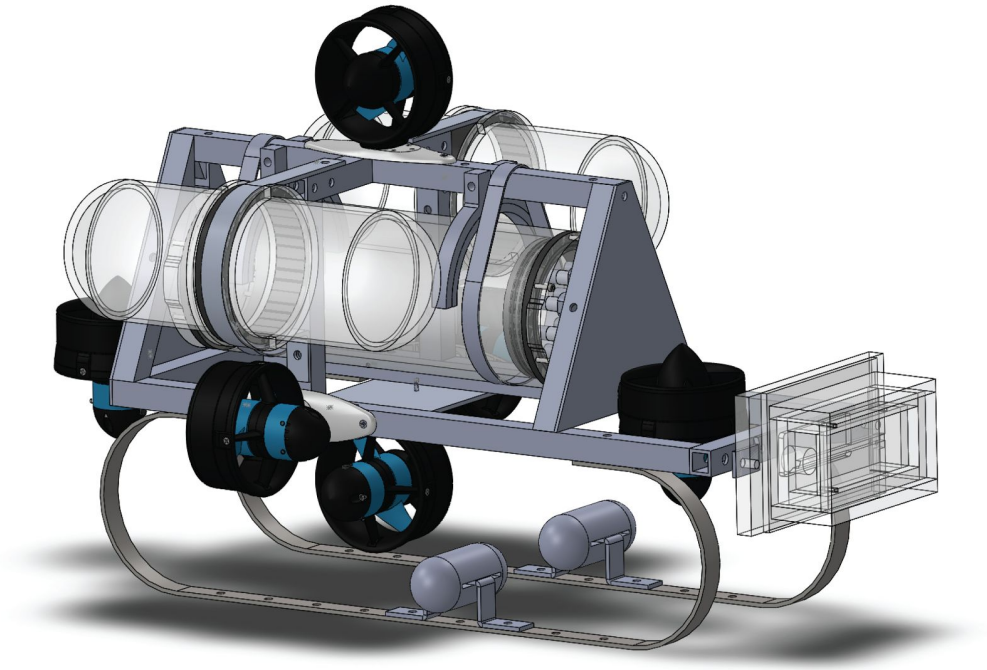


Senior Project:

Control System for an Underwater Remotely Operated Vehicle

Tyler Mau & Joseph Mahoney

Table of Contents

Introduction

Project Overview

The underwater ROV project presents as an open ended endeavor by a large multidisciplinary team. The computer engineering students on the H_2ROV team are working on the overall computer hardware and software design and implementation for the ROV. Our team strives to manually control movement and positioning via a hard-link tether while utilizing computer vision in order to track the ROV's position based on visual references.

Our team broke up the project into smaller reasonable goals. The initial task involved establishing motor control with a PID control system as well as implementing some sort of user control station. We had to integrate a camera with UI components and implement computer vision algorithms with OpenCV to detect underwater references. We wanted to have a basic understanding and implementation of these project aspects and various underwater sensors before the physical ROV prototype was completed. We created a test rig that places motors in a similar orientation as the final vehicle in order to test the PID correction when the rig is tilted off its axes. Once the mechanical engineering team finishes the first physical prototype, we will be ready to test our implementation on the ROV. The ROV will initially be controlled via a direct link to a laptop with a front facing HD camera which feeds video via ethernet to a surface ground station for viewing. After this basic functionality is achieved, we will implement algorithms for the ROV to be able to locate underwater features and stay in place with small counteracting motor thrusts.

The ROV provides a great platform to monitor various underwater environments. Information obtained from an underwater ROV familiarizes the operators with unchartered territory. The H_2ROV team strives to make the task at hand achievable through careful design and implementation processes. The practicality of the ROV motivates this team toward the end

goal. Lastly, this platform provides the base for all future groups to build upon in the Northrop Grumman Collaboration Project.

Clients and Community Partners

The underwater ROV project is sponsored by Northrop Grumman Corporation, a leading contractor for the US Department of Defense, with work on the first generation of the ROV beginning in Fall 2015. One of Northrop Grumman's main business areas is focused in the development of unmanned systems for the military, including unmanned aerial and underwater vehicles. This ROV is part of a larger Northrop Grumman sponsored UAV Collaborative project, which includes several generations of UAVs developed by the Cal Poly SLO and Cal Poly Pomona student teams. The ultimate goal of collaboration with the UAV team is to establish a means of communication between the UAVs in flight and the ROV underwater to accomplish a task to be defined in the future. The underwater ROV is being built in collaboration with a group of Mechanical Engineering students focused on the physical vehicle. This project will be delivered to the client as well as Cal Poly for future development.

Stakeholders

This project has many different stakeholders. First and foremost, because the immediate future of our ROV lies with a future group, next year's Capstone students are our first stakeholders. For that reason, we need to make sure we construct the ROV so that future teams can improve upon it, rather than needing to start over from scratch and thus wasting time. Examples of other stakeholders include scientists, such as environmentalists or marine biologists. These individuals are considered stakeholders because they should be able to use the ROV for underwater data collection via the onboard sensors. If the system does not support functionality that the user needs, they should be able to implement those features without too much trouble.

Framed Insights and Opportunities

Since the scope of this project in the long term is large, we are tasked with defining the needs of the client. Northrop Grumman has given us a clean slate to work with; they want to give us the means to produce a large scale project. The large project ideal stems not only from the complexity of the problem (underwater robotic navigation) but also from the open ended nature of the project. In this project, there are multiple teams from multiple engineering factions that must coordinate plans and actions as one cohesive unit. Ideally, Northrop Grumman wants this team to make an autonomous underwater ROV that can not only communicate with some sort of ground station, but with UAVs flying above or near the ROV as well. Although the communication between the two vehicles is the task of a future team working on this project, it is in our best interest to pave the way as much as possible. This will secure future funding from our client, Northrop Grumman, for future student teams.

In addition to our client, we also have another stakeholder: the Mechanical Engineering (ME) team. Our team constantly coordinates with the ME group to brainstorm and design the ROV system together. In order to be successful, we must constantly communicate with them to ensure that our software system interfaces well with the physical design of the ROV. Most of our communication with the ME team has been about the motor selection, motor placement, and camera options/placement. We must remain aware of the ME team's design decisions, as they may need software implementations to run certain subsystems. The coordination with the ME team is an interesting and realistic opportunity for us to emulate what often occurs in industry. At this point, both this group of CPEs and the ME team have started to formulate a good dynamic. The ME team checks with us often, regarding their ideas and if we can work with their design decisions. Both teams participated in a productive dialogue about crucial design decisions regarding the motor placement. We chose to have motors on both the x and y axes of the ROV for forward and lateral movements as well as motors on the top for pitch and altitude adjustment. We had to compromise between the ease of design for the ME team and our goal of maneuverability with respect to the software implementation side.

Our client liaison between Northrop Grumman and this team, Professor Slivovsky, has been integral in terms of providing advice and showing us how to move forward with certain tasks. She has left the project mostly under our control providing us with an overview of what is expected of the team. One key suggestion that was made was to purchase an OpenROV kit. Professor Slivovsky emphasized that it may be a good idea to do this in order to be able to write code and test our implementations on an actual working ROV while waiting on the manufacture of the ME team's design. Another benefit for ordering the kit was to become familiar with a similar system and understand the challenges associated with underwater navigation and implementation of such systems. The only main hindrance was the price of the OpenROV kit, but we decided that this would be a smart decision, regardless of the price, so the kit was ordered. Studying the OpenROV's design gave us great insight into the considerations that must be made when designing this type of system, so purchasing the kit was definitely beneficial. In the end, our electronic control system was modeled after the choices made by the OpenROV project.

Overall, we have a large group with a lot of talent along with great funding from Northrop Grumman, so this project has been a great learning experience.

Project Goals and Objectives

Our group established multiple goals in order to work toward the expectations of our client, Northrop Grumman. Considering use cases for our platform, we specified several development paths in order to meet these requirements. In order to meet basic movement and positioning requirements, our first objective was to implement manual motor control via a hard-link tether to a control station. Another goal was for the system to acquire and filter sensor information. In our final implementation, we decided to use the same inertial measurement unit sensor that was provided with the OpenROV kit. This sensor provided us with accurate angular positioning data (discussed further in the "Final Design" section below). In order to move towards the goal of autonomous functionality of the ROV, it is necessary to track the vehicle's relative positioning. Thus, within the scope of the senior project, which will continue through spring

quarter, our goal for autonomous functionality is to stabilize lateral positioning using computer vision. However, some stabilization is possible using the IMU only. To achieve this, we implemented a PID control system to stabilize the angular position of the ROV by applying thrust to compensate for positional error. Our goals for the end of Senior Project included basic functionality of the prototype ROV with good maneuverability via the ground station control. We also planned on focusing our development on additional features that could be implemented on future vehicle iterations such as computer vision. We have achieved a successful first prototype of our goals and will be passing off our work to future groups for further development.

Project Outcomes and Deliverables

Through discussions with the mechanical engineers, it has been decided that we would like to deliver a system that can function as follows:

- The ROV shall be able to be moved and deployed by one person.*
- The ROV shall be able to be controlled with any laptop computer.
- The ROV shall be remote-controlled via tether to a control station.
- The ROV shall be able to laterally stabilize itself within 1 meter from reference.*
- The ROV shall be able to communicate sensor data to the ground station via tether.
- The ROV shall have a video latency of up to 0.1 seconds.
- The ROV shall weigh no more than 30 lbs.*
- The ROV shall be operational at depths up to 10 meters below the surface.*
- The ROV shall be waterproofed to IPX8 standard.*

* **NOTE**: These are dependent on the prototype being developed by the mechanical engineering team which is still in progress.

An ROV with the above capabilities will be delivered as the result of our work and serve as a solid platform for future students to develop upon. Specifically, at the end of Capstone II, we aimed for the ROV to be controlled from the control station laptop, transmit a clear video feed,

communicate sensor data in an efficient manner, and stabilize itself underwater. These have been determined based on our conversations with the client, Northrop Grumman, the team of mechanical engineers, and with other professors on campus. We have discussed the scope of the project, and all group members agree that limiting the scope to these outcomes is the best way to ensure success in both the short term and the long term. We aimed to perfect the most essential functions of the ROV so that it not only worked, but so future students would feel completely confident in building more advanced features upon this platform as well, thus ultimately creating a platform that our stakeholders and partners can work with.

Background

Taking part in a newly introduced project to Cal Poly, our team did not have any design documentation or source code from past groups to reference. While this was an exciting opportunity that allowed us to define the project ourselves, it also posed a challenge since we had no established basis for development. However, we discovered a great startup company that provides an open source underwater ROV for consumers to purchase and further develop upon. The company, OpenROV, provides ROV kits that contain all of the parts needed to build their product. The OpenROV website provides a ton of information about the product, and is also an open source community with forums and git repositories for the OpenROV source code. Our team ordered one of the adventure kits which includes extra development materials as well as an entire ROV to assemble. The OpenROV kit provided us with an opportunity to explore various design decisions and considerations that were made in the development of a refined product, such as the choice of motors and camera, two key aspects of our design discussion. From a software perspective, the OpenROV utilizes the BeagleBone Black and Arduino development boards as flexible and powerful platforms with dozens of input and output channels. Mirroring this choice allows us to accurately and efficiently obtain sensor data and communicate between the control station and ROV. Important to note, however, is the fact that the OpenROV kit is not intended as a platform upon which expansive hardware and

software developments can be made. Its open nature supports some development, but the resulting overall design is consumer-oriented, especially with respect to the minimal physical system it incorporates. While the OpenROV allows users to explore underwater environments, we have developed a platform that can be used for much more.

Engineering Specifications

Our goal was to implement the hardware and software systems necessary to produce an operational underwater ROV with the ability to maneuver responsively at the user's command, stream sensor information and a video feed for the user's viewing with low latency, and remain in equilibrium while underwater by accounting for drift. These follow from our desire to create a platform that excels in performing the essential functions of an underwater ROV - one that can support much greater functionality as future developers improve upon it.

The project is directed toward four main end users (actors) in the use case model: a future capstone group at Cal Poly, a Northrop Grumman Engineer, a Navy Technician, and a marine scientist. Each of these actors has similar roles regarding their interaction with the ROV, but each also has different overall goals. As we took the needs of each of these different users into account, we found that any dissimilar needs would be satisfied through additions made to the flexible platform that we planned to develop. As an example, if a marine scientist were interested in sampling soil at the ocean floor, a motorized arm would be needed to take the sample. An improvement such as this was not included in the scope of our project, but we considered it and other potential additions by making the platform as flexible as possible. The engineering requirements that we set reflect our goal of accommodating the needs shared by all of our end users. The result is a versatile system that can easily be built upon as future developers seek to meet the multitude of specific requirements that could be placed on it.

The use cases that we expect our end users to place on the system are as follows: (see **Appendix E** for diagram view)

1. The ROV shall maneuver underwater in response to commands from a control station (laptop) on land.
2. The ROV shall transmit a video feed back to the control station.
3. The ROV shall gather, filter, and display data from the following sensors: accelerometer, gyroscope, magnetometer, pressure sensor, distance/proximity sensor.
4. The ROV shall be able to perform autonomous movement. The drift error correction is being implemented in the scope of our Capstone project as the foundation for future development in this area.
5. The ROV being developed in the scope of our Capstone project shall provide a platform for future development of additional features.

The following engineering specifications were derived from the use cases of the end users as defined above and the customer and engineering requirements defined in **Appendix F.** The Req. # column of the table below refers to the engineering or customer requirements that the specification is derived from.

H2ROV Formal Engineering Requirements

Spec. #	Parameter Description	Requirement or Target	Tolerance	System Risk*	Attainability Risk**	Compliance	Req. #
1	Speed	2 knots	Min	L	L	A, T, S	C1, C7
2	Camera refresh rate	30 Hz	Min	L	L	A, T	C9, E5
3	Sensor sampling rate	1ms	±0.05ms	L	L	A, T, I	C3, E4
4	Weight	30 lbs.	Max	M	M	A, T, I	C7, C8, E6
5	Waterproof	IPX8 Standard	Min	H	M	A, S	C6, E1, E3
6	Net Buoyancy	+3 lb	± 0.2 lb.	M	M	A, T	C6, E1
7	Drift error in any direction	1.5m	± 1m	H	H	T, I	C2, C3, C4, E2, E8, E9
8	Operating Range	100m	Min	L	M	T, S, I	C5, C7, E1
9	Operating Temperature	20 °C	± 20 °C	M	L	A, S	C4, C6, E1

Key:

Compliance - Analysis (A), Test (T), Similarity to Existing Designs (S), Inspection (I)

Risk - High (H), Medium (M), Low (L)

* The System Risk is defined to be the consequences to the system should the requirements not be met, with high risk being catastrophic to the system and low risk being an inconvenience to the user.

** The Attainability Risk is defined to be the risk of the specification not being met, with a high risk being unlikely to be fully realized in the scope of the Capstone project and a low risk being completely realistic. This metric was developed in the project's planning stages but still holds value after the project's completion time as an indicator of complexity of the design - whether completed or incomplete.

Final Design

Design Overview

When designing the vehicle, our group took into consideration the most important aspects of an operational underwater ROV. We decided that precise motion control of the vehicle and a few other key features (see “Key Features” below) were our main priorities. Over the duration of two academic quarters, we discussed various concepts regarding the physical design of the ROV, electronic hardware systems, and software design. This project is sponsored by Northrop Grumman and thus, we took care in ensuring that our design would not exceed our allowed spending budget. (See “Budgets and Expenses” for a cost breakdown.)

Key Features

We designed our vehicle to have a set of key features based on customer requirements and our self-defined concept of an fully-operational underwater ROV. On top of its most basic functionality, these are the features currently implemented on our vehicle that were developed based on our engineering requirements:

User Control Station

- Browser-Based Graphical User Interface
- Manual Motion Control via Keyboard Interface
- Live Video Feed

Control System

- PID Control System
- Seafloor Feature Tracking
- Angular, Lateral, and Vertical Motion Stabilization

Physical Design

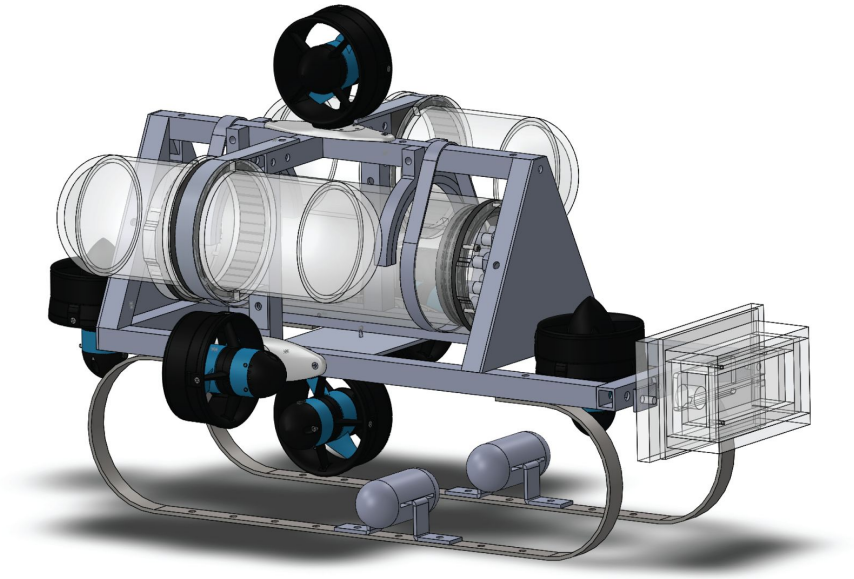


Figure 1: ROV Prototype Design Model

Although the mechanical engineering team guided the physical development of the vehicle, our group specified the desired placement of the thrusters that would give us optimal lateral, vertical, and angular control from a software development standpoint. Through research and discussion, we decided on a six-thruster design (**Figure 1**) which maximizes maneuverability without convoluting the software implementation. The ROV will have pairs of thrusters mounted such that movement is allowed through all six degrees of freedom (though the user is restricted from dictating the vehicle's roll angle). Each pair of thrusters will be symmetric about the center of mass of the vehicle to simplify our control implementation. One of our final design goals involves autonomous lateral stabilization with respect to a reference point underwater; thus, it is imperative that our ROV is able to make fine movement adjustments in the horizontal plane. Maneuverability of the ROV is limited by the physical design, so it was important to discuss this aspect with the mechanical engineering team. The mechanical engineers are currently in the final development stages for the physical prototype.

Physics

When designing a control system, it is important to consider the physical design of the vehicle in order for obtain desired real-world physical reactions based on the software implementation. Due to our vehicle design and thruster placement, rotation about a single axis can affect thrust vectors on another. Thus, to compensate for the changes in forces in our control system, we model our vehicle as follows:

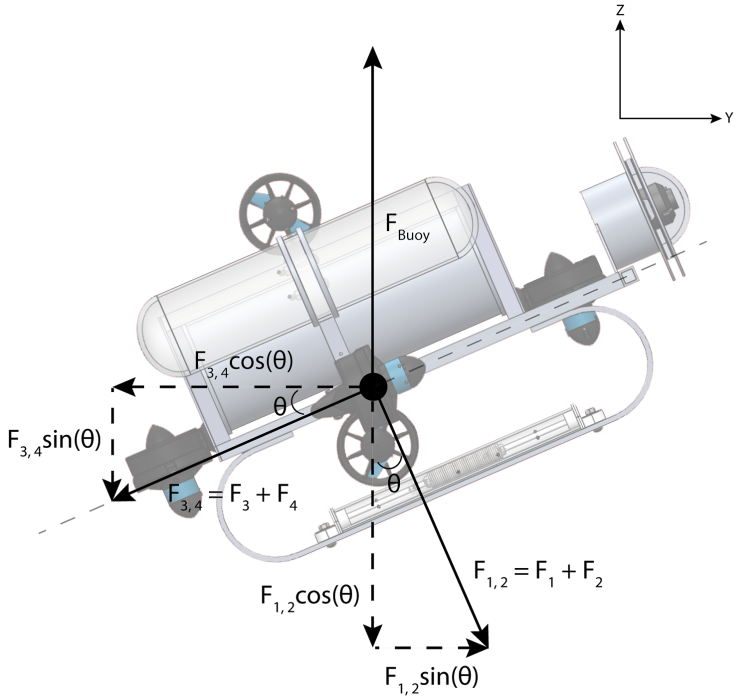


Figure 2: ROV Thrust Vector Modeling

F_1 and **F_2** represent the forces from the **vertical** thrusters

F_3 and **F_4** represent the forces from the **horizontal** thrusters

Our control system allows the user to manually control the pitch of the vehicle, causing the vertical thruster forces to lose magnitude in the vertical direction while gaining a horizontal thrust vector. Without compensation, this would cause the vehicle to drift both vertically and laterally due to the buoyancy force and gained horizontal thrust. Thus, we use thrust vector

modeling (**Figure 2**) in order to calculate new thrust values for the vertical and horizontal thrusters in order to compensate for these drift issues.

To calculate the necessary horizontal thruster values to compensate for lateral drift, we relate the horizontal thrust vectors of both the vertical ($\mathbf{F_{1,2}}$) and horizontal ($\mathbf{F_{3,4}}$) thrusters:

$$F_{1,2}\sin(\theta) = F_{3,4}\cos(\theta)$$

$$F_{3,4} = F_{1,2}\tan(\theta) \qquad \textbf{Eq. 1}$$

In order to compensate for vertical drift, we relate the buoyancy of the vehicle to the vertical thrust vectors of the vertical and horizontal thrusters:

$$F_{Buoy} = F_{3,4}\sin(\theta) + F_{1,2}\cos(\theta) \qquad \textbf{Eq. 2}$$

Thus, using the above equations (**Eq. 1** and **Eq. 2**), we are able to solve for the necessary vertical thruster value to compensate for vertical drift:

$$F_{Buoy} = F_{1,2}\tan(\theta)\sin(\theta) + F_{1,2}\cos(\theta)$$

$$F_{1,2} = F_{Buoy} / [\tan(\theta)\sin(\theta) + \cos(\theta)] \qquad \textbf{Eq. 3}$$

Solving equations **Eq. 1** and **Eq. 3** will produce the thrust values for both the horizontal and vertical thrusters that compensate for the lateral and vertical drift.

Thrusters

Figure 3: BlueRobotics T100 Thruster

(https://www.bluerobotics.com/wp-content/uploads/2014/08/blueesc-11.png)

The thrusters we decided to use on the ROV were pre-built and designed for underwater applications. In choosing these thrusters (**Figure 3**), we solved a few concerns that we discussed during the design phase:

- The built-in electronic speed controllers (ESCs) reduce the concern of excess heat in our capsule since they will no longer be placed in the central containment unit.
- Due to the thrusters being designed for underwater applications, we no longer have to worry about carefully waterproofing motors to work as underwater thrusters.

The thrusters also provided a few added benefits:

- The thruster design allows for easy mounting onto our vehicle.
- BlueRobotics provided CAD models to help with the physical prototype modeling.
- BlueRobotics performed extensive testing on the thrusters and provided corresponding performance charts.
- The thrusters are simple to use and come with thorough documentation.

Thrust to PWM

The thrusters discussed above are operated by sending pulse-width modulation (PWM) values to the built-in ESCs. The thrust of an individual thruster is dependent on its rotational speed, which is determined by the received PWM values. Therefore, once performing calculations to obtain the necessary thrust values to run each of the thrusters at, we must convert those thrust values to PWM values in order to actuate the thrusters accordingly. To accomplish this, we analyze the thrust-PWM relationship provided by BlueRobotics. This relationship has been verified through their extensive testing.

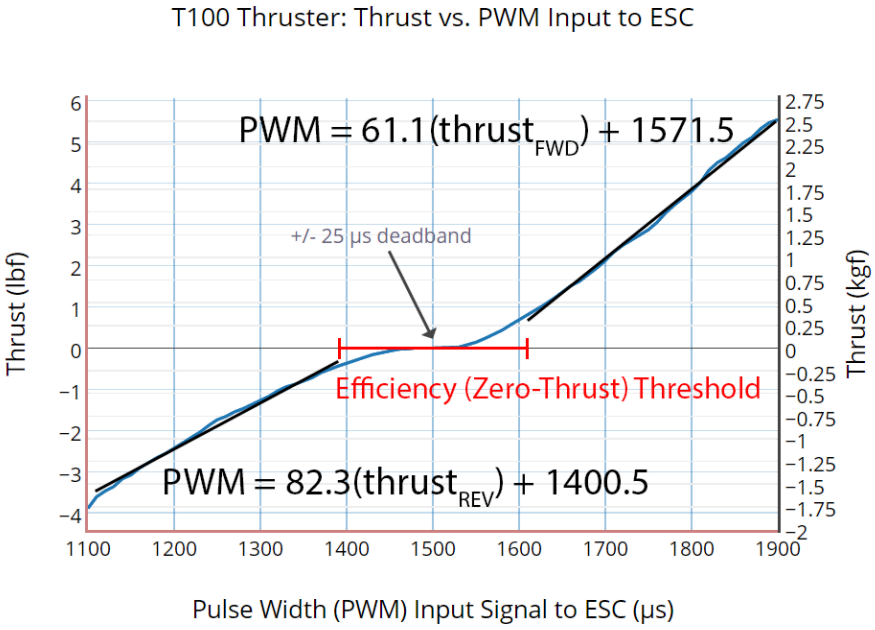


Figure 4: Thrust to PWM Approximations

We could not simply design our control system based on PWM values due to the nonlinear and nonsymmetric forward and reverse thrust-PWM relationships. Since our thruster placements are designed to be symmetric about the vehicle's center of mass, optimal control is obtained through symmetric forward/reverse thrust values. Therefore, using the above graph (**Figure 4**), we estimated linear relationships for both the forward and reverse directions to actuate the thrusters at approximately the correct speed.

Although our linear approximations are reasonably accurate within their range (shown as the black lines on **Figure 4**), we were left with a small range that could not be modeled linearly (labeled as the "efficiency threshold"). After some discussion, we determined it was not necessary to model the relationship within this range for several reasons:

- Considering the mass of the vehicle, thrust values within this range are insignificant.
- Within this range, the thrusters operate at their lowest efficiency.
- Thrust values within this range would only be necessary for correcting very small angular errors. However, the PID controller requires minimum angular error values to prevent the thrusters from having high runtime within the lowest-efficiency range.

Thus, the thrusters will not run within the "efficiency threshold" and values within the range will be set to zero.

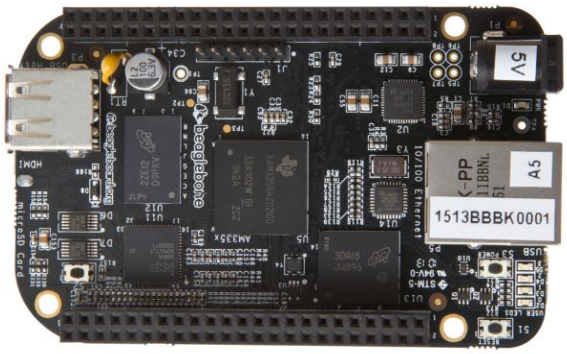


Figure 5: BeagleBone Black Microcomputer
(http://infinetix.com/wp/wordpress/wp-content/uploads/2014/02/projects_bbb.jpeg)

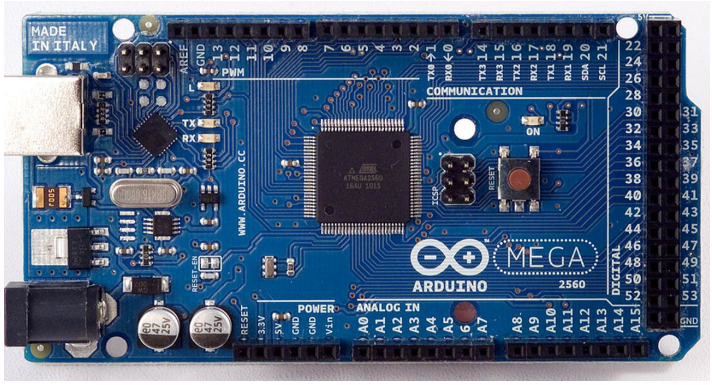


Figure 6: Arduino Mega 2560 Development Board
(https://hifiduino.files.wordpress.com/2012/03/arduinomega2560_r2_front.jpg)

Our system's electronic hardware consists of two main components: the BeagleBone Black microcomputer (**Figure 5**) and the Arduino Mega 2560 development board **(Figure 6**). These two components work together as a completely functional ROV system.

The **BeagleBone Black** is a microcomputer running a Debian Linux distribution with a 1 GHz processor. Due to its processing power, the BeagleBone Black is responsible for the following system functionalities:

- Hosting a Node.js web-server to allow communication via ethernet tether.
- Handling keyboard input for user commands through a browser-based user interface.
- Packaging and sending user input data to the Arduino Mega via a UART serial interface.
- Sending low-latency live video feed from the on-board camera to the browser client.
- Running computer vision algorithms for seafloor tracking.

The **Arduino Mega 2560** development board contains an ATmega2560 microcontroller with a processing speed of 16 MHz. Due to our team's familiarity with AVR architecture and the Arduino's extensive libraries for our required data communication protocols, we utilize the Arduino for the following system functionalities:

- Reading inertial measurement unit (IMU) sensor data to calculate setpoints and errors for use by the PID controller.
- Parsing received user command data sent from the BeagleBone Black via UART.
- Packaging and sending sensor data back to the BeagleBone Black via UART.
- Running the PID controller to calculate correction values for stabilizing the vehicle's angular position.
- Actuating thrusters based on error correction values from the PID controller and parsed user command data.

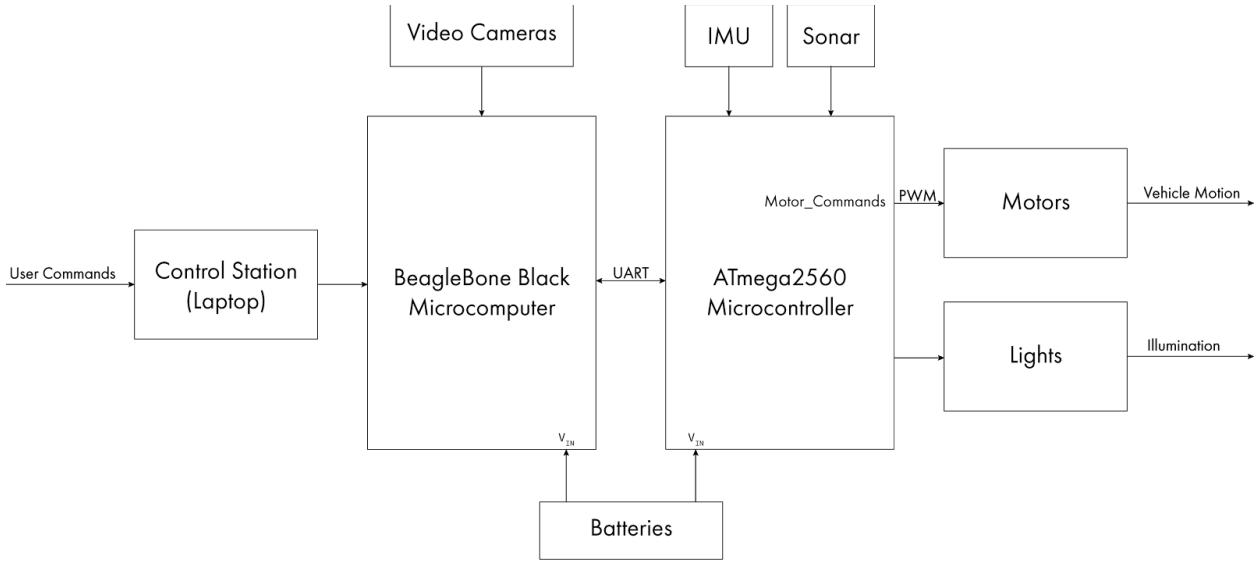


Figure 7: High-Level Electronic System Flow Diagram

Figure 7 shows a high-level diagram of our electronic system. See **Appendix B** for a detailed schematic of the entire system.

Software Design

Overview

We categorized our software design into four main components. These components were developed and integrated together to create a fully operational ROV system.

- User Control Station
  - User interface for manual control
  - Developed on BeagleBone Black
  - Node.js server backend
- Communication Protocol
  - Bit-vector design for UART serial communication
  - Developed on BeagleBone Black and Arduino Mega
- Control System
  - PID control system for angular stabilization
  - Developed on Arduino Mega
- Computer Vision
  - Approximates distance traveled via seafloor tracking
  - Developed using OpenCV on BeagleBone Black

User Control Station

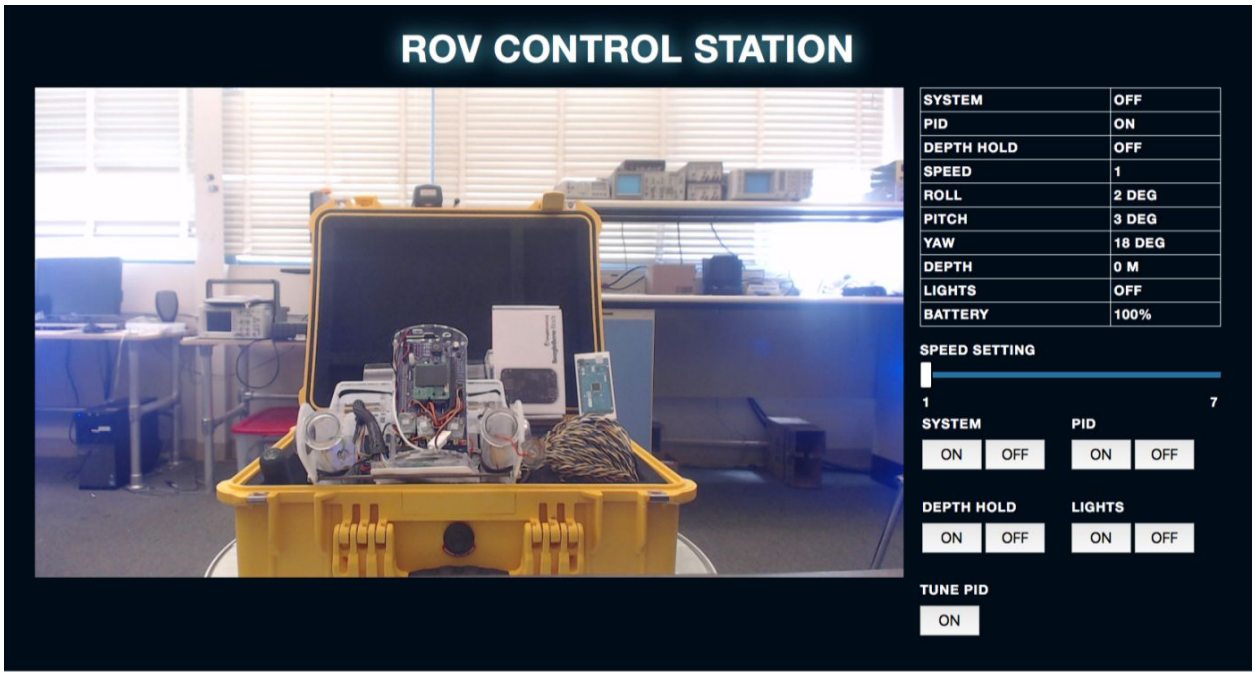


Figure 8: Browser-Based User Control Interface

The user control station is a browser-based interface that allows the user to manually control the ROV's motion via keyboard commands. The following commands are supported:

Key Pressed	Motion Command
Up Arrow	Lateral Move Forward
Down Arrow	Lateral Move Backward
Left Arrow	Lateral Move Left
Right Arrow	Lateral Move Right
W	Pitch Up
S	Pitch Down
A	Yaw Left (Counter-Clockwise)
D	Yaw Right (Clockwise)
Shift	Dive
Spacebar	Ascend

Table 1: Keyboard Commands for Manual Control

Additionally, the user interface (**Figure 8**) includes features that allow the user to specify a speed setting and control the onboard lights. More features will be implemented in the future.

Node.js Server

The browser control station provides an easy way for the user to quickly connect to the device and begin controlling it without installing any software. This is made possible by the websocket library called socket.io. The Beaglebone Black hosts a Node.js server as a backend that handles user input from the browser interface. The webpage is stored on the BeagleBone Black and is served to the browser when the user visits http://192.168.7.2:8888, which is the server's IP address. Below (**Figure 9**), we observe this relationship more clearly. The Node server, after listening for user input, is able to control the BeagleBone's GPIO pins to send data serially to the Arduino.

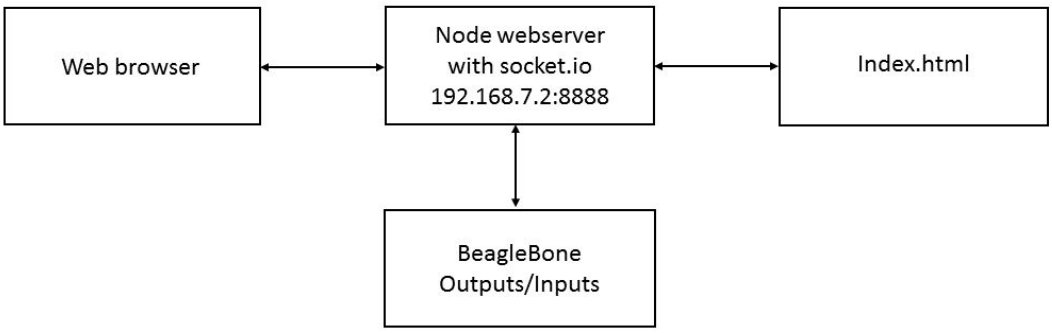


Figure 9: High Level Socket.io Relationships

The socket.io libary allows for real-time synchronous and bidirectional communcation between the UI and the Node server. This is important to ensure low latency responses from the BeagleBone when sending data to the Arduino. The server constantly listens for user input and handles keyboard and mouse click events. This event driven approach works well for constant input changes. There is a theoretical "tunnel" between the browser and the server which is

always open and connected. The system can remain running and respond accordingly to all inputs in any order or time period.

In a sample run (**Figure 10)** of a "forward" arrow key command event, we can observe the chain of events that occur to handle this user interaction properly. On the client side, the up arrow is pressed and a group of functions handle that request and pass proper data to the socket.emit() function. This function is a socket.io API call that is responsible for sending information through the "tunnel." On the server side, the socket.on() API call is responsible for listening for events being sent. This listening or sending of events can be done on either the client or server side. On the BeagleBone server side, a group of functions handle the forward command and prepare the bit vector packet to be sent serially to the Arduino for motor actuation.

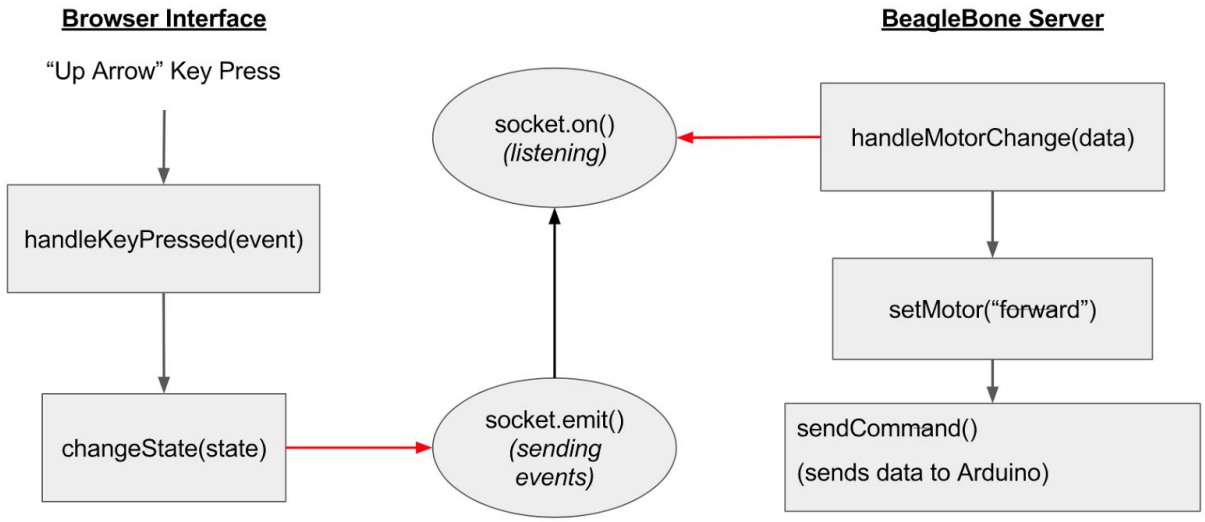


Figure 10: Sample Socket.io Event Handling

Communication Protocol

In order to send user command data from the BeagleBone Black to the Arduino Mega, we package data into a custom four-byte bit vector. This allows us to send easily-parseable data over the UART serial protocol. Our bit vector design is shown below (**Figure 11**).

31 - 15	14	13 - 11	10 - 1	0
N/A	Lights	Speed	Motion Commands (See Below)	Parity

10	9	8	7	6	5	4	3	2	1
Yaw Left	Yaw Right	Pitch Down	Pitch Up	Left	Right	Fwd	Back	Up	Down

Figure 11: Four-Byte Bit Vector Design

The data is packaged on the BeagleBone Black through JavaScript functions and is parsed on the Arduino side through our self-designed and implemented API (C++). For simplistic error checking, we implemented a parity set/check to verify the validity of data when parsed on the Arduino side. Furthermore, our communication protocol also performs sanity checks to ensure contradicting commands are not being sent simultaneously (e.g. forward and backward).

In the future, we will design and implement a similar communication protocol for sending sensor data from the Arduino to the BeagleBone Black.

Note: The data must be sent through a logic level converter since the BeagleBone Black operates at 3.3V while the Arduino operates at 5V. This is shown in the detailed system schematic in **Appendix D**.

Control System

In order to have precise motion control over our ROV, it is important to know the orientation and heading of the vehicle in 3-dimensional space. To obtain this information, we utilize an **inertial measurement unit** (IMU) which consists of an accelerometer, gyroscope, and tri-axial magnetometer. Each of these individual components have their advantages and disadvantages.

The **accelerometer** allows us to determine the orientation (tilt) of the vehicle by outputting acceleration readings in g-forces on all three axes with respect to Earth's gravitational pull. However, since the accelerometer measures gravitational forces, any external forces, including small disturbances or vibrations, will affect the device's output. Therefore, to accurately determine the vehicle's orientation, it is not enough to use the accelerometer exclusively.

In order to compensate for the accelerometer's sensitivity to external forces, we utilize a **three-axis gyroscope**. A gyroscope outputs data in the form of angular velocities (in degrees) on all axes. Since the device measures the change in angles over time, it is not prone to linear disturbances like the accelerometer, and therefore is more accurate on a smaller scale. However, gyroscopes are known for two main issues: gyroscopic bias and drift. Gyroscopic bias simply means that when the device is static, the gyroscope outputs a consistent, non-zero value. This can be hard-code adjusted when reading data from the sensor, but the bias usually changes during operation due to shifts in temperature. Thus, it is better to deal with the bias in a filtering algorithm. Gyroscopic drift is inherent to most consumer-grade gyroscopes which causes the output values to drift over time. However, the accelerometer data won't drift over time, so it's often used as a reference to compensate for the gyroscopic drift.

While the accelerometer and gyroscope are useful for determining the orientation of the vehicle with respect to the x and y axes, they do not provide enough information to know the heading (rotation about the z-axis) of the vehicle. To obtain this data, we use a **tri-axial magnetometer**, which measures magnetic strength (in milliGauss) on all three axes. The goal of this device is to utilize Earth's magnetic field as a reference point to determine the change in

heading of the vehicle. However, this device is prone to magnetic distortions, which are categorized into hard-iron and soft-iron biases. Hard-iron biases occur in the presence of permanent magnetic fields, such as Earth's magnetic field. Depending on your location in the world, the magnetic strength will differ and calibration of the magnetometer is necessary. We achieved calibration by graphing the sensor's output data on a three-dimensional graph and adjusting the output values by a bias. The uncalibrated magnetometer readings can be seen in **Appendix A** and the calibrated output in **Appendices B and C**. Soft-iron biases occur when ferrous metals get close enough to distort the sensor's readings. These distortions are usually taken care of in a filtering algorithm.

Sensor	Advantages	Disadvantages
Accelerometer	- Stable over time	- Sensitive to vibrations/disturbances
Gyroscope	- Not prone to linear disturbances - Accurate readings over short periods	- Values drift over time - Gyroscopic bias
Magnetometer	- Stable readings	- Prone to magnetic distortion - Hard-iron bias - Soft-iron bias

Table 2: Summary of Advantages and Disadvantages of IMU Components

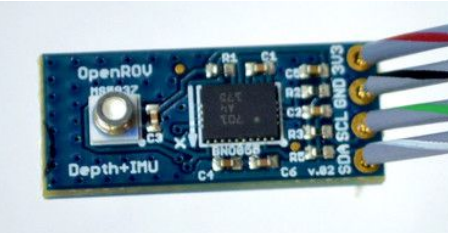
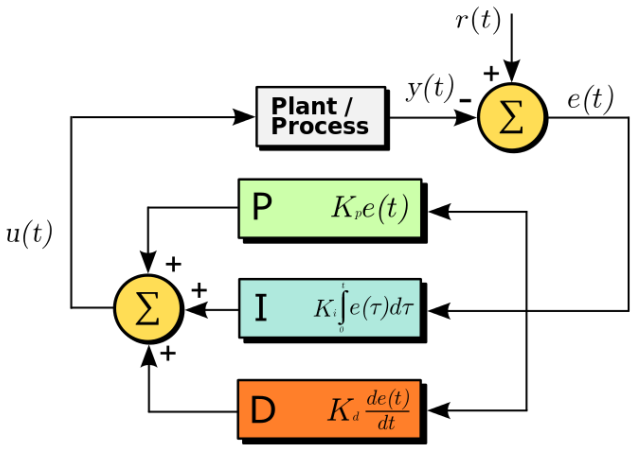


Figure 12: IMU + Pressure Module

(https://cdn.shopify.com/s/files/1/0200/0264/products/DSC_1226_1024x1024.JPG?v=1437523168)

In order to utilize all three IMU components to compensate for each of their disadvantages, we need to filter and fuse the data to combine the advantages of each sensor to provide stable and accurate positioning data. The IMU we decided to use, the BNO055 9-axis orientation sensor (**Figure 12**), performs sensor data fusion onboard the chip and outputs data in the form of scaled euler angles. This provides us with accurate and responsive data, allowing us to save computational time from having to implement sensor fusion algorithms ourselves.



(https://en.wikipedia.org/wiki/PID_controller)

Figure 13: PID Controller Diagram

After determining the orientation and heading of the ROV, the next step is to actuate thrusters in such a way that the vehicle positions itself to the desired location. To accomplish this, we use a PID controller (**Figure 13**), a popular control algorithm in mobile systems. PID stands for Proportional, Integral, Derivative -- the three main components of the algorithm. All three parts of the algorithm are combined into an error correction term, which is then used to actuate the thrusters to achieve a desired position over time.

- The **proportional** part of the controller will contribute to error correction directly based on the amount of current error.

- The **integral** aspect will consider the accumulation of error over time to compensate for error offsets.
- The **derivative** portion of the controller accounts for the change in error over time, which helps reduce system overshoot and oscillations.

The error values used in these calculations are based on the IMU data and desired angular position (based on user commands).

However, the common implementation technique for the derivative part of the controller can cause a problem known as **derivative kick**. When the setpoint (desired position) is changed, the error changes almost instantaneously which causes spikes in the derivative term of the controller. However, by performing derivative on measurement, as opposed to derivative on error, we still account for the change in error over time, but we neglect the instantaneous changes in setpoint. This is because the change in error can be defined as:

$$\text{Change in Error} = (\text{Change in Setpoint}) - (\text{Change in Measurement})$$

Derivative kick only comes from the change in setpoint, which affects the change in error only when the setpoint is changed. Therefore, by removing the setpoint from the equation, we are left with:

$$\text{Change in Error} = -(\text{Change in Measurement})$$

In a system that requires fine/detailed positioning, we cannot allow major spikes in error correction that could cause erratic vehicle motion.

After combining the proportional, integral, and derivative terms, we are left with a correction value (in lbs of thrust) for a single axis of motion (e.g. y-axis/pitch). Thus, we need at least three total PID controller components in order to have full control over all three axes.

After calculating thrust correction values for each axis of motion, we combine these values with user command data to obtain final thrust values for each thruster. These values are then converted to PWM values (as discussed in a previous section) to actuate the thrusters accordingly. At this point, the control loop restarts by calculating new setpoints and error values. See **Figure 14** at the end of this section for a detailed software flow diagram.

Computer Vision

While GPS is often used for absolute localization, underwater vehicles cannot utilize this luxury. This is due to the physical limitations of GPS, where electromagnetic radiation loses power when traveling through the water. Alternatively, systems that measure distance using linear acceleration readings lack the accuracy necessary for autonomous navigation. Some solutions available can provide accurate distance translation data, yet can cost thousands of dollars each and require large amounts of power to operate, which is often not available on small underwater vehicles. Therefore, alternate methods, such as the feature-tracking method described above, must be explored to achieve real-time distance information.

In order to implement autonomous capability on our ROV, we have two main goals:

- Achieve autonomous lateral stabilization over a fixed ground reference
- Approximate distance traveled via seafloor tracking and distance sensing

To achieve these goals, the implementation needed to be developed so that it could run on the vehicle's microcomputer (BeagleBone Black), which runs a Debian Linux distribution. Therefore, the project was implemented using OpenCV which has C++ libraries that are compatible with Linux operating systems. A downward-facing camera on the ROV is used to detect and track features on the seafloor.

The implementation begins by connecting to a webcam and storing a video frame to be processed. This frame is converted to grayscale and sent as a parameter to OpenCV's built in FAST algorithm function. After the first set of features are detected on the first frame, another frame is obtained from the camera to begin the feature tracking. The current frame, previous frame, and feature points are sent to OpenCV's KLT tracking function which produces a set of new points that were tracked from the previous frame. Using the old and new locations of the tracked feature points, an affine transform is estimated using another function provided by OpenCV's libraries. This function helps remove any falsely tracked points that could potentially skew the data. After obtaining an affine transform matrix from the function, the translation information is used to estimate pixel distance traveled in the x and y axes and are added to a total distance sum (for each axis) and outputs those values to the terminal.

From this point, the program loops, constantly capturing frames and performing feature tracking until there less than the minimum number of points required for tracking (specified by the user). Once below this minimum number of tracked points, a new set of feature points are obtained through the FAST algorithm and the tracking begins again. Below is a high-level software flow diagram of the implementation (**Figure 14**).

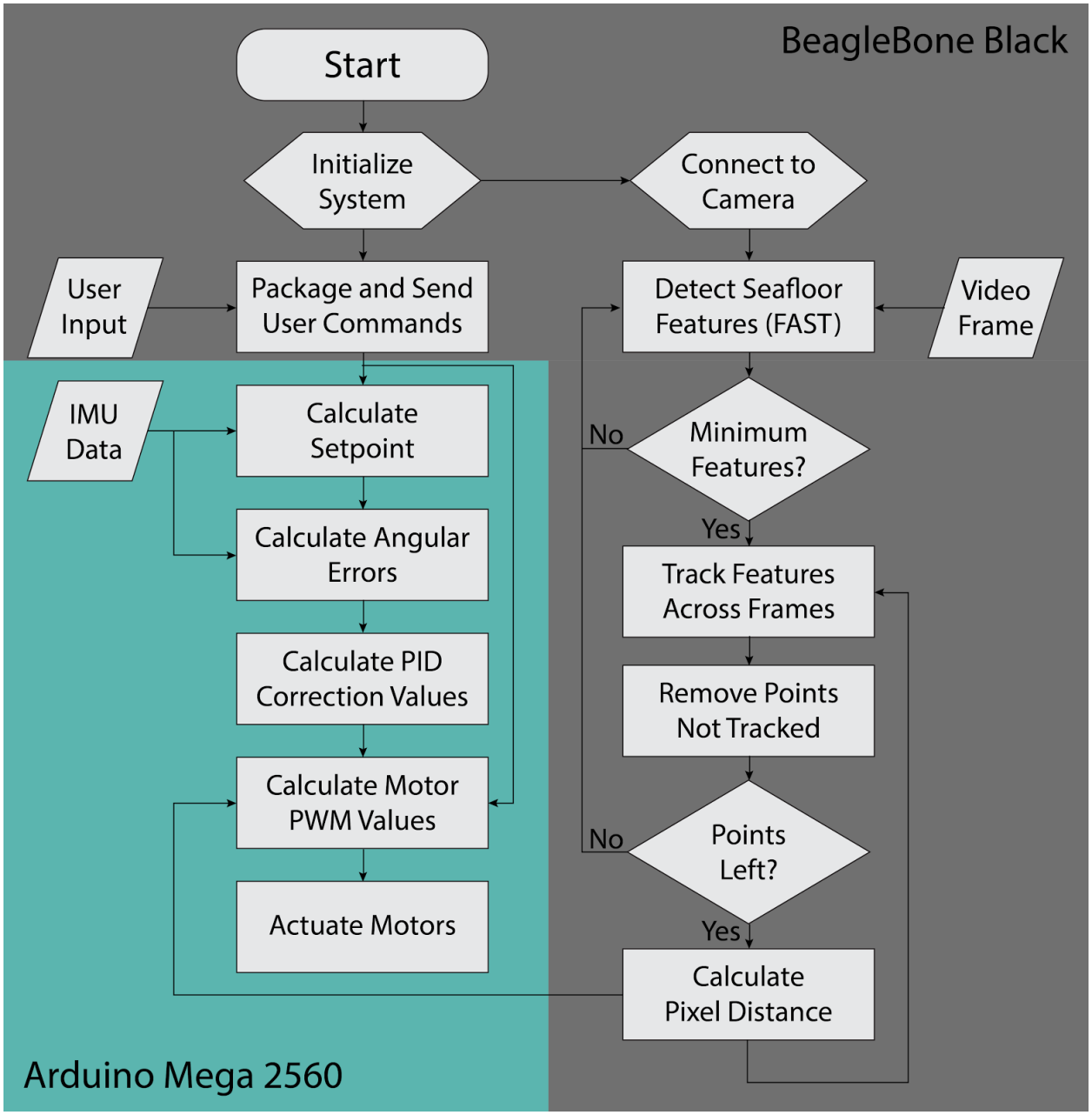


Figure 14: ROV Software Flow Diagram

However, the results are skewed when there are moving objects in the frame because the points move along with the moving object(s). In our project, we are working on fixing this problem by eliminating the moving object(s) from the frames to remove the skewed points. In order to do this, we use the BackgroundSubtractorMOG2 method provided in the OpenCV library to extract the moving object(s). It's a Gaussian mixture-based background/foreground segmentation algorithm that selects the appropriate Gaussian distributions for each pixel, which provides better adaptability to varying illumination between scenes and also detects shadows. The BackgroundSubtractorMOG2 method generates a binary mask of the moving object(s). We invert the generated mask so that the moving objects are black (0 in pixel value), so that we can bitwise AND the mask with the original frame to generate the original frame without the moving object(s). We realized that the generated frame leaves some holes in the detected objects, leaving blotchy patches of where the objects are located. Consequently, we explore the use of blurring and morphological operations on the mask to fill in the holes for a better masked frame.

Gaussian blur yielded better results, but did not get rid of all the holes completely by itself. Blurring the mask a few more times completely removes the moving objects, however, the masked area becomes large. Since the feature tracking algorithm only uses a few points for its calculations, it isn't an issue. For future applications like object tracking, we would like to refine the mask. We move on to explore morphological transformations. We used dilation which dilates an image by using a given structuring element, to join the broken parts of the objects. The mask created after dilation to fully rid the image of moving object(s) is better than relying solely on blurring, but also covers a lot of the image when there are many moving objects. We also tried using image closing which is also useful in closing small holes in objects. This method results in a slightly more refined mask, but still result in a lot of masked pixels when there are many objects present. The morphology shapes available to generate the structuring element for the morphological operation cannot maintain the integrity of the original object(s) while filling in the holes. The original generated mask with less intensive image transformations would probably be better for object detection.

Computer Vision Results

Overall, the system works as expected. Given a live video feed, the program displays detected features on the resulting video display and outputs the total pixel displacement to the terminal window. The following describes the test procedure:

1. Position computer to a known starting location
2. Begin implementation code
3. Move the computer/camera slowly in one direction, observing the pixel displacements output to the screen in real time
4. Stop moving the computer at a known "stop" marker
5. Move the computer slowly in the reverse direction, back toward the start point
6. Stop moving the computer when it reaches its initial starting point

When the camera reaches the initial starting position, a net distance of zero is expected.

This testing procedure determined both the accuracy and speed of the distance-tracking system. **Figures 15 and 16** show the starting and ending frames of our test. The final displacement results can be seen in **Figure 17** below.

Figure 15: Starting Frame

Figure 16: Final Frame

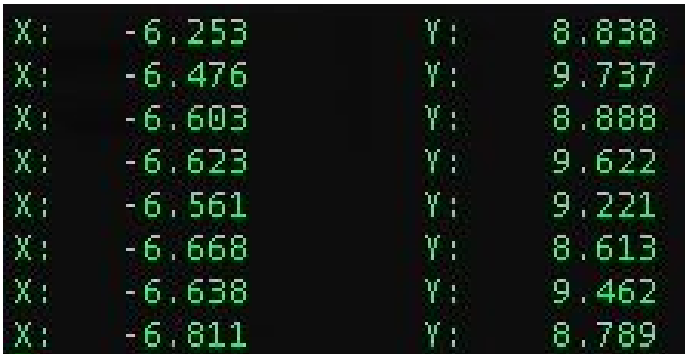


Figure 17: Console Output of Final X and Y Pixel Displacement

A video demonstration can be found at:

https://www.youtube.com/watch?v=Bz1A2JmBoKo

Unfortunately, underwater video was unable to be obtained for testing the implementation. Testing was restricted to above-water video, which is reasonable given this is the first prototype implementation. However, it would be beneficial to further develop the algorithm using underwater video, since computer vision implementations are often driven by their applications.

System Integration and Testing

Failure Mode and Effects Analysis (FMEA) Overview

Failure mode and effects analysis was conducted to determine high risk factors in the system. A full version of the FMEA spreadsheet can be found in **Appendix G.** It was found that the highest area of concern was tether entanglement which could cause damage to the motors or obstruct motor actuation. The risk priority number (RPN) of the tether becoming entangled is quite high, especially when considering poor tether management and the unlikelihood of detection before failure. Therefore, a tether spooling system is being researched to promote better tether management and avoid entanglement. Another significant high risk area of concern is water leakage which may be caused by poor water sealing or cracks due to trauma. Despite occurrence being relatively low, the presence of water inside the system would be catastrophic to the electronics and cause total failure. Detection by the user would also be fairly difficult until post-catastrophic failure. Water leakages can be avoided through careful submergence tests and checking for cracks or damage prior to each field test. We can help increase detectability by implementing water sensors within the containment unit. Several other failure risks were also identified, but were given lower priority consideration due to their low risk priority number.

Design Verification Plan and Report (DVP+R) and Analysis

A full copy of the Design Verification Plan and Report spreadsheet can be found in **Appendix H**. Due to time constraints and the physical prototype delay, many of the test cases outlined in the DVP+R spreadsheet were not able to be completed. Two of the design verification tests performed and completed are outlined below. In both instances, the system was able to consistently pass the tests.

Test 1: T100 Thrust Measurement Test

Determine: Real-world underwater motor thrust values given PWM input to the ESCs

Materials: PVC Pipe Testing Rig, T100 Motor, Arduino Mega 2560, Battery, Computer

Safety: Don't put hands near the thruster, make sure thruster is submerged, keep electronics away from water

Procedure:

1. Attach and secure T100 thruster to end of wooden testing plank.
2. Place wooden plank onto metal rods connected to middle of the PVC test rig.
    a. Secure wooden plank from lateral movement.
    b. Ensure thruster is completely submerged underwater
3. Secure thrust scale to top of PVC rig and connect to wooden test plank.
4. Connect the Arduino to the computer and run the test program.
5. Connect thruster to the Arduino and battery.
6. Send thrust values through the serial communication window.
    a. Send values in increments of 0.2 lbs of thrust.
    b. Measure and record corresponding values from thrust scale.
7. Flip thruster on test plank to test the reverse thrust.
8. Repeat steps 1 through 7 for other thrusters for complete testing.

Test 2: Command Passing Test

Determine: Whether the user keyboard command is properly parsed to the Arduino

Materials: BeagleBone Black, voltage level shifter, Arduino Mega 2560, breadboard, wire leads

Safety: Ensure wires are connected correctly to prevent electrical failures

Procedure:

1. Connect the BeagleBone and Arduino serially via the UART TX/RX pins but make sure that the level shifter is connected in between the two boards.

2. Press every keyboard command once.
3. Monitor the serial COM port from the Arduino IDE .
4. Verify that each command was parsed correctly by the Arduino and ready for further work.
5. Repeat with different command combinations and with multiple commands at once.

Overall System Analysis

Of the five requirements that we defined for this project, three were met. Below is a restatement of these requirements coupled with summaries of progress for each.

1. The ROV shall maneuver underwater in response to commands from a control station (laptop) on land.
    - To the extent possible without a physical ROV prototype completed by the team of mechanical engineers, this requirement was met. Our control station reliably sends commands to the motors.
2. The ROV shall transmit a video feed back to the control station.
    - This requirement was met fully. Software running on the Beaglebone Black processes camera frames and streams them to the browser with low latency. A resolution of 1280x720 was achieved with a smooth frame rate of 30 frames per second.
3. The ROV shall gather, filter, and display data from the following sensors: accelerometer, gyroscope, magnetometer, pressure sensor, distance/proximity sensor.
    - This requirement was partially met. The Arduino Mega successfully gathers acceleration, angular velocity, and heading data, making use of them in the control system. However, these values are not yet displayed to the user through the control station user interface. Research was done on distance sensing, but this feature has also yet to be implemented.

4. The ROV shall be able to perform autonomous movement. The drift error correction is being implemented in the scope of our Capstone project as the foundation for future development in this area.
    - This requirement was partially met. Feedback from the IMU factors into the control system's stabilization of the ROV. Full lateral stabilization will require integration of the completed computer vision system being developed by the senior project group.

5. The ROV being developed in the scope of our Capstone project shall provide a platform for future development of additional features.
    - This requirement was met. Our system is straightforward and well-documented. In terms of the mechanical, hardware, and software systems, our implementations are easy to build upon in many ways.

Management Plan

The success of this project hinged on our ability as a team to coordinate the completion of multiple development tasks in parallel. Once the major aspects of the ROV were developed independently, the process of integrating them together commenced. Our team used the agile methodology to facilitate this process. Using aspects of the Scrum framework, we organized all tasks by their level of completeness, including testing status. This gave us an excellent understanding not only of our overall progress, but also of the project's dependencies and flow. We further organized tasks by importance, allowing us to set priorities and plan ahead.

Independent Development Phase

Three key areas were independently developed:

- **Control System** - Development of a codebase for reading sensor data and actuating motors for vehicle stabilization

- **User Control Station** - Browser-based user interface and research into integrating a camera with the BeagleBone Black
- **BeagleBone Black / Arduino Interfacing** - Developing a serial communication protocol for packaging and sending user commands from the BeagleBone Black to the Arduino via UART.

Integration Phase

The next integration phase consisted of the combination and interfacing of the ROV's major components. Predominantly, this included the development of unified codebases on the BeagleBone Black and Arduino that supported the functionality developed in the previous phase while allowing complete system-wide operation.

Budget and Expenses

Cost Breakdown

Physical Prototype Design/Fabrication	$1800.00
Arduino Mega 2560	$45.95
BeagleBone Black Rev C	$55.00
Alibi ALI-IPV3113R Dome Security Camera	$119.99
Wires/Miscellaneous Parts	$40.00
Inertial Measurement Unit (IMU)	$199.00
OpenROV Adventure Kit	$1388.50

Budget Totals

Total Budget	$5000.00
Total Estimated Cost	$3648.44
Excess Budget	$1351.56

The Learning Experience

This project provided a platform for learning that would be difficult to match. We practiced all types of engineering philosophy from ideation, design, and implementation to testing, review, and redesign. It was also necessary for our designs to be compatible with the physical designs of the vehicle by the mechanical engineering team. The system integration alone was a huge task in this project. But like everything with engineering, all details must be accounted for. Creating the software design and implementation was a hefty task but one thing that was overlooked in the project was the actual assembly of the electronics onto the ROV. We greatly underestimated how difficult this task would be. Almost all aspects of the vehicle and software were thought out except for this piece. It took us about one week to adequately wire the motors through the end cap to the containment unit. Waterproofing everything would also prove to be a challenge. The way the cable penetrators worked was that they were a intermediary to the inside of the containment unit but not a seal. This meant that the wires would have to be epoxied into the end caps which would create a scenario where no changes could be made once the epoxy was set. Upon finishing the wiring of the vehicle we were fitting all of the electronics into the rather small containment unit for a test fitting, we caused a spark while plugging in the battery that simultaneously caused all six motors to fail and thus become unusable.

Although this accident happened at the end of the quarter, the future students who work on this project will be guided by this mistake and not to overlook any single detail of the design process. Adding a larger containment unit and a fuse would be a good start to solve this issue. With our robust software system, the next group of students who work on the ROV will be able to get the vehicle in the water and verify its capabilities.

References

ID	Information Gathered	Source
1	T100 Thruster Information	https://www.bluerobotics.com/store/thrusters/t100-thruster/
2	OpenCV Tracking Functions	http://docs.opencv.org/2.4/modules/video/doc/motion_analysis_and_object_tracking.html
3	PID Control Basics	http://ctms.engin.umich.edu/CTMS/index.php?example=Introduction&section=ControlPID
4	BNO055 IMU Datasheet	http://www.mouser.com/ds/2/783/BST_BNO055_DS000_13-838248.pdf
5	IMU Code from OpenROV	https://github.com/OpenROV/openrov-software-arduino/blob/master/OpenROV/BNO055.cpp
6	BeagleBone GPIO Pins	http://www.allaboutcircuits.com/projects/how-to-use-the-digital-i-o-on-a-beaglebone/
7	OpenROV Software Architecture	https://github.com/OpenROV/openrov-software/blob/master/docs/OpenRovSoftwareArchitectureOverview.pdf
8	Paper on Autonomous Underwater Navigation	http://www.robots.ox.ac.uk/~mobile/Papers/Robotica.pdf
9	Pressure Sensor Information	http://meas-spec.com/product/pressure/MS5837-30BA.aspx
10	Arduino Library Documentation	https://www.arduino.cc/en/Reference/Libraries
11	Related Written Text (Other Involved Students)	Angela Yoeurng, Andrew Sorensen, Matt Rounds, Jackie Fong

Harris, C. and Stephens, M. 1988. A combined corner and edge detector. In *Fourth Alvey Vision Conference*, Manchester, UK, pp. 147–151.

J. Shi and C. Tomasi, "Good features to track", *Proc. IEEE Computer Society Conf. on Computer Vision and Pattern Recognition*, pp. 593-600, 1994

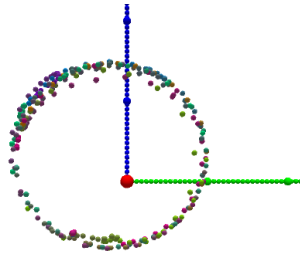
M. Trajkovic and M. Hedley. Fast corner detection. Image and Vision Computing, 16, 1998.

G. Farnebäck. Two-frame motion estimation based on polynomial expansion. In Scandinavian Conference on Image Analysis, 2003.

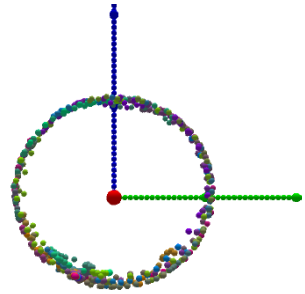
J. Suhr. Kanade-lucas-tomasi (klt) feature tracker. Technical report, Computer Vision (EEE6503) Course , Yonsei Univ., 2009.

Appendix

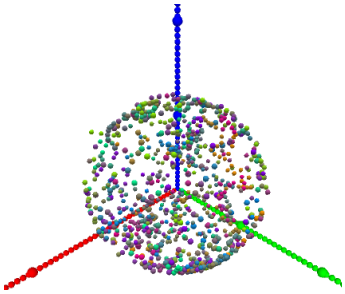
A. Uncalibrated Magnetometer (x-axis)



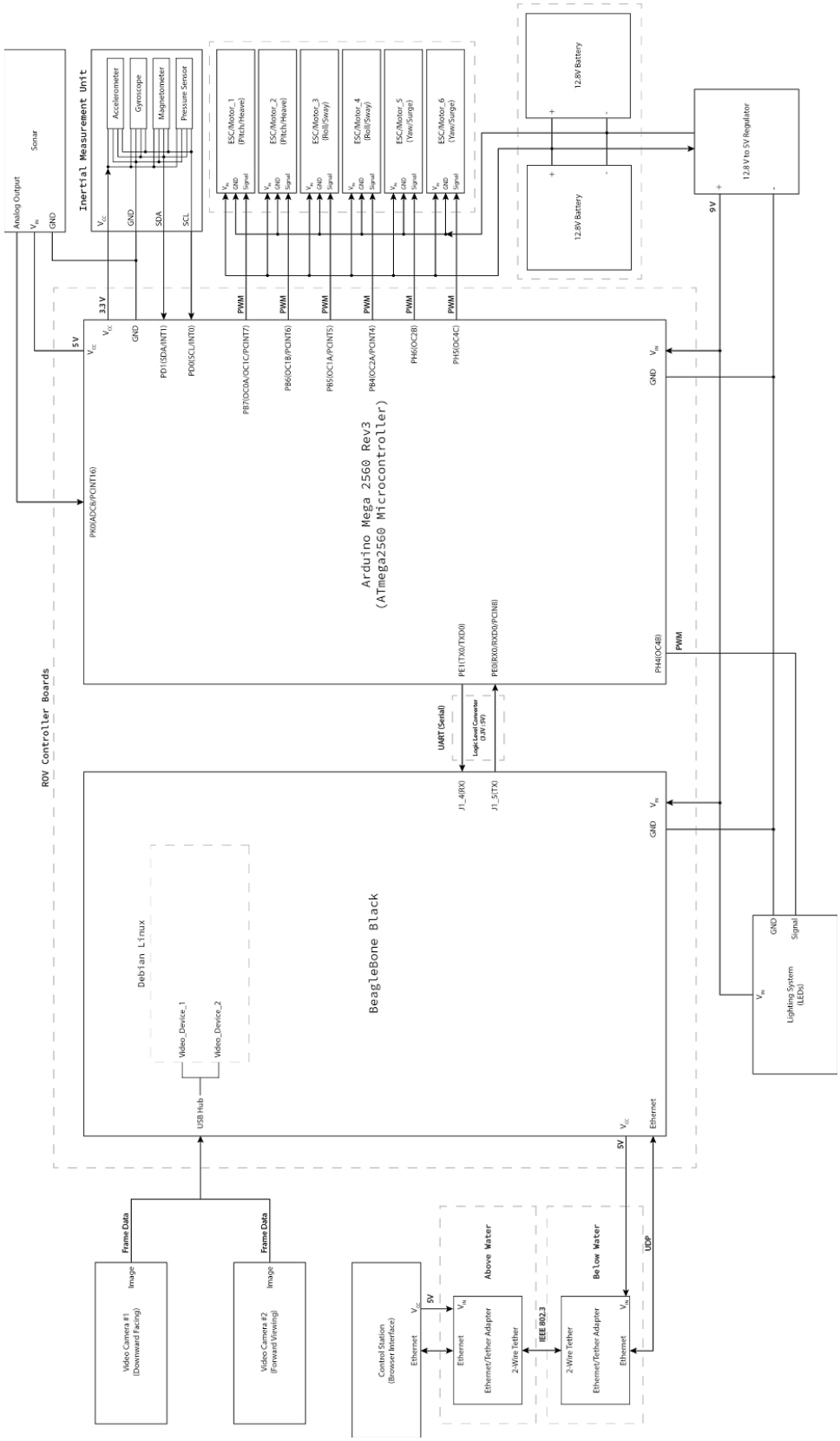
B. Calibrated Magnetometer (x-axis)



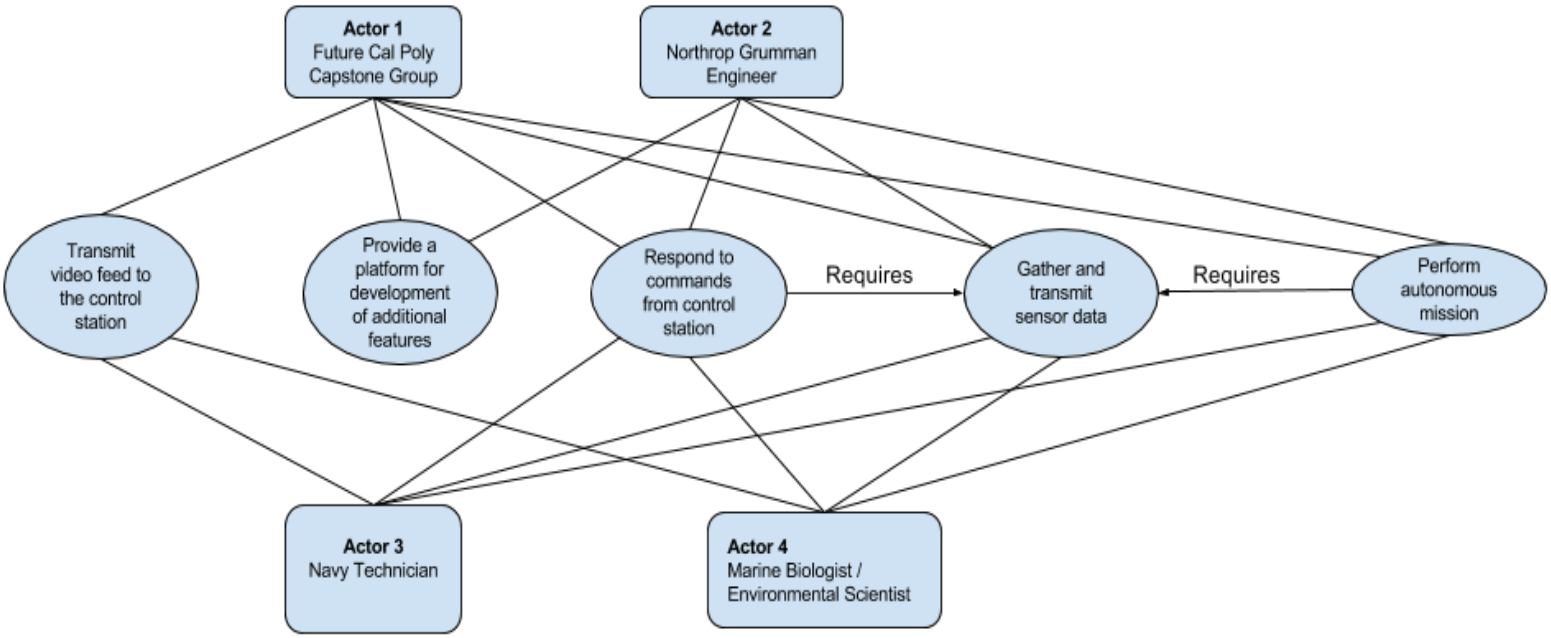
C. Calibrated Magnetometer (all axes)



D. Detailed System Schematic



E. Use Cases Diagram



F. Customer and Engineering Requirements

Req. #	Customer Requirements
C1	System shall have remote-controlled movement capabilities through a laptop
C2	System shall have some autonomous capability
C3	System shall accurately measure positioning
C4	System shall be physically stable in water
C5	System shall be able to communicate to laptop
C6	System shall be able to dive underwater
C7	System shall be highly maneuverable
C8	System shall be able to be deployed easily
C9	System shall send video data back to a laptop with minimal delay

Req. #	Engineering Requirements
E1	System shall operate at depths up to 10 meters and be able to rise back to the surface on command
E2	System shall be able to stay in fixed location (accounting for drift) within 1 meters from original point of reference
E3	System shall be waterproof to IPX8 standard
E4	System shall transmit sensor data
E5	Video feed shall have a refresh rate of at least 30 Hz
E6	System shall be able to be deployed by no more than one person
E7	System shall have a 1 hour minimum battery lifetime
E8	System shall be able to locate a reference point underwater
E9	System shall be stable from rolling due to underwater currents

G. FMEA

Process Function	Potential Failure Mode	Potential Effects of Failure	Severity	Class	Potential Causes/ Mechanisms of Failure	Occurance	Current Process Control	Detection	RPN	Recommended Actions
Power	Low Power/ Shutdown	Loss of Control/Communication Damaged Batteries	7		Failure to Charge Battery	2	None	1	14	Implement battery voltage monitor and automatic shut-off.
Submersion	Water Leak	Damaged Electronics Sinking	10		Bad Sealing Cracks	2	Visual Inspection	6	120	Create immersion test environment before adding electronics.
	Excess Pressure	Cracking or Damage to Vehicle Potential Water Leaks	10		Travel Too Deep Using Weak Materials	2	Pressure Sensor	6	120	Perform pressure tests and disallow vehicle from exceeding recommended depth.
Control	Stabilization Failure	Loss of Control Undesired Movement/Orientation	8		Bad Algorithms Motor/ESC Failure Controller Failure	4	IMU Sensor Algorithms	3	96	Implement algorithm to detect and correct exceedingly erratic motion.
Motor Actuation	Cavitation	Undesired Movement Damaged Propellers	6		Motor Speed Too High	1	Known PWM Control	6	36	Limit max speed of motors.
	Tether Entanglement	Movement Limited Damaged Propellers	8		Bad Cable Management	5	None	8	320	Create tether management spooling system.
	ESC Failure	Loss of Control	7		Overuse Wrong Input Voltage	2	None	8	112	Ensure proper connection of ESCs.
	Loss of Balance	Loss of Control	3		Bad Algorithms	2	IMU Control Algorithms	2	12	Adjust PID parameters.
	Frozen Process State	Loss of Control "Dead State" MOTORS WILL CONTINUE	8		Infinite Loops	2	Timeouts for Waits	5	80	Ensure proper timeouts are implemented for any waits.
Communication	Lost Connection	No Manual Control	8		Tether Breaking Wireless Failure Control Station Failure	3	None	1	24	Add visual to control station to indicate lost connection. Ensure good connection. Implement automatic surfacing.
Distance Sensing	Sonar/Laser Inaccuracies	Inaccurate Stabilization Object Avoidance Failure	2		Physics Inaccurate Equipment Bad Algorithms	7	None	5	70	Implement dual measurement system for improved accuracy at closer distances.
Vision	Water Clouding	Vision Tracking Failure	2		Bad Water Conditions	8	None	5	80	Dual lighting system to prevent particle reflections. Use in clear conditions.
	Incorrect Analysis	Stabilization/Motion Failure	2		Bad Conditions Bad Algorithms	6	None	8	96	Alert user of bad readings and recommend manual control.
	Latency	Control Delay	4		Slow Controller	7	Time Period Calculations	2	56	Write efficient algorithms to reduce latency.

H. DVP+R

CPE 450 DESIGN VERIFICATION PLAN AND REPORT

Report Date		Sponsor			Component/Assembly		REPORTING ENGINEER:	

TEST PLAN

TEST REPORT

Item No	Specification [1]	Test Description [2]	Acceptance Criteria [3]	Test Responsibility [4]	Test Stage [5]	SAMPLES TESTED		TIMING		TEST RESULTS			NOTES
						Quantity	Ty [6]	Start date	Finish dat	Test Res [7]	Quantity Pass	Quantity Fail	
1	Run Time	Compute run time given battery specifications and system power consumption. Theoretical: Run the completed system at full speed until battery voltage reaches unacceptable level	> 20 min.	Matt	CV	1	A	TBD	TBD				
2	Component Depth Rating	Theoretical: Submerge the system for 30 minutes at 33 ft. Repeat.	> 33 ft.		CV	1	A	TBD	TBD				
3	Horizontal Velocity	Calculate velocity given thrust values. Theoretical: Run at full speed past a known distance.	> 0.5 ft./s	Tyler	CV	1	A	TBD	TBD				
4	Video Latency	wave hand in front of camera to observe the response time	< 0.25 s	Joe	PV	10	C	3/7/2016	3/7/2016	Passed	All	none	
5	User Command Latency	Film system response to a key press at 240fps. Review video frame-by-frame.	< 0.1 s	Matt/Joe	DV	1	B	3/11/2016	3/11/2016	Passed	All	none	
6	Buoyancy	Add 4 lb mass to the system and ensure neutral buoyancy.	4 +/- 1 lbf		CV	1	A	TBD	TBD				
7	T100 Thrust Measurement	Test the thrust output of the motor in water given PWM values.	Thrust values consistent with the specs	Tyler	CV	2		2/27/2016	TBD				
8	Command Passing	Sending command bit vector from the BeagleBone to the Arduino and parsing on the Arduino side	All commands parse correctly with no invalid commands	Joe	PV	1	C	2/26/2016	3/5/2016	Passed	All	None	